Introduction to SQL

Class 10

Course Overview

- Introduction to SQL
  - Databases, Tables
  - Classification of SQL – DDL, DML, DCL, TCL
    - DDL – CREATE, ALTER, DROP
    - DML – SELECT, INSERT, UPDATE, DELETE
    - DCL – GRANT, REVOKE
    - TCL – COMMIT, ROLLBACK, SAVEPOINT
  - Data types, Operators
  - Keys – Primary, Foreign, Composite, Unique, Alternate
  - Integrity Constraints – Domain Integrity Constrains, Entity Integrity Constraints, Referential Integrity Constraints
  - Joins – Outer Joins, Left Outer Joins, Right Outer Joins, Inner Joins.
  - Queries, Subqueries, Functions, Flow Control (IF, CASE, WHILE, REPEAT, LOOP), ,Stored functions ,Stored Procedures
  - Views
  - Indexes, Cursors, Triggers, Events
  - Concurrency and locking (Implicit locks, explicit locks, row level locks, table level locks, database level locks)
  - Tuning SQL queries and optimizing performance
  - SQL Databases vs NoSQL Databases
  - ACID, CAP
  - How SQL databases internally works

Stored Functions

- Stored functions in MySQL are used to encapsulate a set of SQL statements into a single, reusable routine (function). They can be used in a variety of situations, including:
  - To perform complex calculations or data manipulations that need to be reused throughout the application.
  - To encapsulate business logic or validation rules that need to be applied consistently across multiple parts of the application.
  - To simplify the application's SQL code by abstracting away complex or repetitive queries.
  - To improve performance by allowing the database to optimize the execution of the stored function.
  - To improve security by allowing the database to execute certain privileged operations without giving the application direct access to the underlying tables.
  - To make the database more modular by breaking it down into smaller, reusable components.

Stored Functions

- Characteristics
  - Stored functions return a single value
  - CREATE ROUTINE database privilege is needed to create a stored function

```
DELIMITER $$

CREATE FUNCTION fun_name(fun_parameter(s))
RETURNS datatype
[NOT] {Characteristics}
fun_body;
```

Returns all the routines from travel database

```
SELECT * FROM information_schema.routines
WHERE routine_schema = 'travel';
```

Stored Functions

- A stored function is a user-defined function that can be used in SQL statements.
- A stored function is defined using the CREATE FUNCTION statement and can take one or more input parameters and return a single value or no value.
- Stored functions can contain SQL statements, control flow statements, and variable assignments.
- Stored functions are similar to stored procedures, but they return a value whereas procedures do not.
- A stored function can be called from a SELECT statement, a SET statement, or from within another stored function or stored procedure.
- Stored functions can be used to encapsulate complex business logic or data validation rules.
- Stored functions are precompiled, which means that the database server performs some optimization on the function when it is created, making its execution faster.
- Stored functions can be used in various contexts, including SELECT, UPDATE, DELETE and INSERT statements.
- Stored functions can be recursive, meaning that a function can call itself, to a certain level.
- Stored functions can be called from triggers, events and views.

Stored Functions

Parameter Name	Descriptions
fun_name	It is the name of the stored function that we want to create in a database. It should not be the same as the built-in function name of MySQL.
fun_parameter	It contains the list of parameters used by the function body. It does not allow to specify IN, OUT, INOUT parameters.
datatype	It is a data type of return value of the function. It should any valid MySQL data type.
characteristics	The CREATE FUNCTION statement only accepted when the characteristics (DETERMINISTIC, NO SQL, or READS SQL DATA) are defined in the declaration.
fun_body	This parameter has a set of SQL statements to perform the operations. It requires at least one RETURN statement. When the return statement is executed, the function will be terminated automatically. The function body is given below: BEGIN -- SQL statements END $$ DELIMITER

Stored Functions - Examples

A function to calculate the total cost of a purchase order:

```
CREATE FUNCTION get_total_cost(order_id INT)
RETURNS DECIMAL(10,2)
BEGIN
  DECLARE total_cost DECIMAL(10,2);
  SELECT SUM(price * quantity) INTO total_cost
  FROM order_items
  WHERE order_id = order_id;
  RETURN total_cost;
END;
```

Stored Functions - Examples

A function to check if an email address is valid:

```
CREATE FUNCTION is_valid_email(email VARCHAR(255))
RETURNS BOOLEAN
BEGIN
  DECLARE email_regex VARCHAR(255) DEFAULT '^[a-zA-Z0-9._%+-
]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$';
  RETURN email REGEXP email_regex;
END;
```

Stored Functions - Examples

A function to calculate the average salary of employees in a department:

```
CREATE FUNCTION get_avg_salary(department_id INT)
RETURNS DECIMAL(10,2)
BEGIN
  DECLARE avg_salary DECIMAL(10,2);
  SELECT AVG(salary) INTO avg_salary
  FROM employees
  WHERE department_id = department_id;
  RETURN avg_salary;
END;
```

Stored Functions - Examples

A function to check if a given date is a weekend or not:

```
CREATE FUNCTION is_weekend(date DATE)
RETURNS BOOLEAN
BEGIN
  RETURN (WEEKDAY(date) = 5 OR WEEKDAY(date) = 6);
END;
```

Stored Functions - Examples

A function to calculate the total number of orders for a customer

```
CREATE FUNCTION get_total_orders(customer_id INT)
RETURNS INT
BEGIN
   RETURN (SELECT COUNT(*) FROM orders WHERE customer_id = customer_id);
END;
```

Stored Functions - Examples

A function to calculate the total number of products in a category:

```
CREATE FUNCTION get_total_products(category_id INT)
RETURNS INT
BEGIN
   RETURN (SELECT COUNT(*) FROM products WHERE category_id = category_id);
END;
```

Stored Functions - Examples

A function to check if a given string is a palindrome or not:

```
CREATE FUNCTION is_palindrome(string VARCHAR(255))
RETURNS BOOLEAN
BEGIN
  RETURN (string = REVERSE(string));
END;
```

Stored Functions - Examples

A function to calculate the total cost of shipping for an order:

```
CREATE FUNCTION get_shipping_cost(order_id INT)
RETURNS DECIMAL(10,2)
BEGIN
   RETURN (SELECT SUM(cost) FROM shipping WHERE order_id = order_id);
END;
```

Stored Functions - Examples

A function to convert lbs to kg

```
DELIMITER //
CREATE FUNCTION lbs_to_kg(lbs MEDIUMINT UNSIGNED)
RETURNS MEDIUMINT UNSIGNED
DETERMINISTIC
BEGIN
  RETURN (lbs * 0.45359237);
END//
DELIMITER ;

SELECT a.plane, max_weight AS max_lbs,
  lbs_to_kg(max_weight) AS max_kg
FROM airplanes a INNER JOIN manufacturers m
  ON a.manufacturer_id = m.manufacturer_id
WHERE m.manufacturer = 'airbus'
ORDER BY a.plane;
```

Stored Functions

- `DROP FUNCTION IF EXISTS lbs_to_kg;`

MySQL IF statements: syntax and examples

- IF-THEN
- IF-THEN-ELSE
- IF-THEN-ELSEIF-ELSE

IF THEN

```
IF condition THEN
  statements;
END IF;
```

MySQL IF-THEN-ELSE statement

```
IF condition THEN
   statements;
ELSE
   else-statements;
END IF;
```

MySQL IF-THEN-ELSEIF-ELSE statement

```
IF condition THEN
   statements;
ELSEIF elseif-condition THEN
   elseif-statements;
...
ELSE
   else-statements;
END IF;
```

MySQL: LOOP Statement

- In MySQL, the LOOP statement is used when you are not sure how many times you want the loop body to execute and you want the loop body to execute at least once.

```
[ label_name: ] LOOP
  {...statements...}
END LOOP [ label_name ];
```

- label_name –
  - Optional. It is a name associated with the LOOP. You use the label_name when executing an ITERATE statement or LEAVE statement.
- You would use a LOOP statement when you are unsure of how many times you want the loop body to execute.
- You can terminate a LOOP statement with either a LEAVE statement or a RETURN statement.

MySQL: LOOP Statement

```
DELIMITER //
CREATE FUNCTION CalcIncome ( starting_value INT )
RETURNS INT
BEGIN
  DECLARE income INT;
  SET income = 0;
  label1: LOOP
    SET income = income + starting_value;
    IF income < 4000 THEN
      ITERATE label1;
    END IF;
    LEAVE label1;
  END LOOP label1;
  RETURN income;
END; //
DELIMITER ;
```

MySQL: REPEAT Statement

- In MySQL, the REPEAT statement is used when you do not know how many times you want the loop body to execute.

[label_name:] REPEAT

{...statements...}

UNTIL condition

END REPEAT [label_name];

- label_name
  - Optional. It is a name associated with the REPEAT loop.
- Statements
  - The statements of code to execute each pass through the REPEAT loop.
- Condition
  - The condition that will terminate the REPEAT loop.
- You would use a REPEAT statement when you are unsure of how many times you want the loop body to execute.
- You terminate a REPEAT statement with the *UNTIL condition*.

MySQL: REPEAT Statement

```
DELIMITER //
CREATE FUNCTION CalcIncome ( starting_value INT )
RETURNS INT
BEGIN
  DECLARE income INT;
  SET income = 0;
  label1: REPEAT
    SET income = income + starting_value;
  UNTIL income >= 4000
  END REPEAT label1;
  RETURN income;
END; //
DELIMITER ;
```

MySQL: WHILE Statement

- In MySQL, the WHILE statement is used when you are not sure how many times you will execute the loop body and the loop body may not execute even once.

[label_name:] WHILE condition DO

  {...statements...}

END WHILE [label_name];

- label_name
  - Optional. It is a name associated with the WHILE loop.
- Condition
  - Te condition is test each pass through the WHILE loop. If the condition evaluates to TRUE, the loop body is executed. If the condition evaluates to FALSE, the WHILE loop is terminated.
- Statements
  - The statements of code to execute each pass through the WHILE loop.
- You would use a WHILE LOOP statement when you are unsure of how many times you want the loop body to execute.
- Since the WHILE condition is evaluated before entering the loop, it is possible that the loop body may not execute even once.

MySQL: WHILE Statement

```
DELIMITER //
CREATE FUNCTION CalcIncome ( starting_value INT )
RETURNS INT
BEGIN
  DECLARE income INT;
  SET income = 0;
  label1: WHILE income <= 3000 DO
    SET income = income + starting_value;
  END WHILE label1;
  RETURN income;
END; //
DELIMITER ;
```

Connect with me on LinkedIn

Please subscribe to our YouTube channel

Check out my GitHub profile

Follow me on Twitter(X)

Thank you